Universidad Carlos III de Madrid

Escuela Politécnica Superior

Grado en Ingeniería de Sistemas Audiovisuales

Desarrollo de un videojuego con Realidad Aumentada para dispositivos móviles Android

Trabajo Fin de Grado

Autor: Israel Matellán Jiménez

Tutor: David Griol Barres

Leganés, septiembre 2016

A mis padres y a mi hermana.

Índice general

Índice de Figuras

Índice de Tablas

Agradecimientos

En primer lugar quiero agradecer a mis padres por ayudarme en todo lo que necesito y estar siempre para cualquier cosa y a mi hermana que aunque a veces no podamos el uno con el otro siempre estarás ahí para lo que necesite. A los yayos y tíos por estar siempre conmigo y apoyarme en todo lo que hago.

Un rincón especial para Lu, que sin tu apoyo y tesón porque siga adelante ha hecho que todos estos años hayan sido más fáciles. ILD.

A mi tutor David por brindarme la oportunidad de realizar este TFG y soportarme durante este periodo.

A todos los compañeros de carrera que han estado presente durante esta andadura que no han sido pocos años.

Por último a los topos (ALM y GBM) que me han ayudado muchísimo en esta última etapa con sus consejos, incluso en el desarrollo de esta memoria. Siempre nos quedarán los mendrus.

Sé que me dejo a gente pero siempre me acordaré de todos vosotros.

¡MUCHAS GRACIAS A TODOS!

Resumen

Los videojuegos han sufrido una gran evolución desde sus inicios en los años 70 hasta la actualidad. Estos avances siempre han ido por el afán de mejorar las tecnologías, haciendo que los costes de nuevas plataformas se vean disminuidos.

La tendencia actual es ir hacia el uso de los dispositivos móviles (Smartphone) como "Gamestation" ya que la gran mayoría de los consumidores de juegos dispone de uno.

El sistema operativo móvil más utilizado es Android por eso los desarrolladores tienden a trabajar con ella y más aún con la salida al mercado de motores de videojuegos compatibles con este sistema como es el caso de Unity 3D.

Una de las tecnologías que están en auge para el entretenimiento es la realidad aumentada, ya que consigue mezclar el entorno real de usuario, con elementos virtuales. Por estas razones son por las que se realiza este trabajo de fin de grado, para poder dar una visión de esta nueva tecnología que es la realidad aumentada con la unión del entretenimiento mediante la creación de un videojuego.

El videojuego que se desarrollará es de tipo "Arcade" que pretende que el usuario termine con los ovnis que van apareciendo en realidad aumentada antes de que se le acaben las vidas.

Palabras clave: Realidad aumentada, Videojuegos, Unity, Vuforia, script, C#, Android, aplicación móvil, target, 3D.

Abstract

Videogames have had a great evolution since their beginning in the seventies to nowadays. The aim of these developments is to improve technologies reducing costs.

Current trend is using mobile devices (Smartphones) as "Gamestation" since the vast majority of gamers have one device.

Due to Android has become the most requested mobile operative system, developers trend to work with it. Even more, with the market the debut of compatible videogame engines with this operative system such as Unity 3D.

Augmented reality is one of the entertainment technologies that is booming because it is able to mix the real environment with virtual items. Because of that, this Final Project has been developed. In order to give a complete vision of augmented reality creating a videogame.

"Arcade" will be the type of videogame to develope. User have to kill UFOs that will be appearing in augmented reality before UFOs kill him.

Key words: Augmented reality, videogames, Unity, Vuforia, Script, c#, mobile application, target, 3D.

CAPÍTULO 1:

INTRODUCCIÓN

1.1 Motivación

La evolución de la tecnología móvil es algo que está presente en el día a día de la sociedad actual. Es una evolución continua en la que cada día se sacan al mercado nuevas aplicaciones para así poder satisfacer la demanda de los usuarios. Estas aplicaciones en su mayoría son videojuegos. Estos videojuegos cada vez necesitan más recursos con lo que prospera con ello que aparezcan nuevos dispositivos móviles con características cada vez más parecidas a las de los ordenadores personales.

Los videojuegos son uno de los mercados que más dinero genera a nivel mundial. Según el estudio de EAE Business School "El mercado del videojuego 2015" en España en 2015 se han invertido 318 millones de euros. Es una cantidad de dinero considerable con lo que no es de esperar que el crecimiento sea tan grande.

Uno de los problemas que se encuentran los desarrolladores es plantearse hacia que plataforma orientarse. El crecimiento de los videojuegos en dispositivos móviles es palpable, ya que cada vez más la gente se encuentra fuera de sus domicilios así que la única forma es que jueguen con dispositivos móviles como los Smartphone.

Por otro lado, cada vez más las empresas están interesadas en tecnologías como la Realidad Aumentada (RA) o la Realidad Virtual (RV). Con la RA se consigue combinar el mundo real que envuelve al usuario con el virtual añadiendo para ello elementos tridimensionales.

Como la RA está en auge, y se puede aplicar a cualquier dispositivo que tenga una cámara con la que reconocer los elementos y una pantalla, una de las mejores plataformas que se adaptan a ello es Unity3D ya que permite exportar el juego o aplicación creados a diversas plataformas como Android, iOS, XBOX, PC…

1.2 Objetivos del proyecto

Con este trabajo de fin de grado lo que se pretende es realizar un videojuego utilizando para ello la tecnología de Realidad Aumentada usando para ello la plataforma de desarrollo Unity3D [1].

Para ello se darán las pautas seguidas durante el desarrollo del videojuego sin ser un manual de dicha plataforma. El videojuego se denomina UFO ATTACK!. Los puntos del videojuego que se tratarán son los siguientes:

- Preparación del entorno de desarrollo.
- Diseño de un modelo 3D.
- Implementación de scripts para realizar las funcionalidades necesarias.
- Creación de un sistema de puntuación y vidas.
- Añadir sonido.
- Exportar el videojuego.

Adicionalmente se expondrá el sistema operativo usado para el desarrollo de este videojuego Android, los motores de desarrollo de videojuegos, la evolución de los videojuegos y de realidad aumentada.

1.3 Gestión del proyecto

Para la realización de la planificación se ha utilizado un listado de tareas que se adjunta a continuación:

Tarea

Nombre	Fecha de inicio	Fecha de fin
Creación TFG	9/05/16	21/09/16
Estudio previo	9/05/16	15/07/16
Definición del poyecto	9/05/16	11/05/16
Información Unity 3D	11/05/16	15/06/16
Aprendizaje C#	10/06/16	8/07/16
Aprendizaje 3Ds Max	8/07/16	17/07/16
Realización del proyecto	18/07/16	15/09/16
Definición del videojuego	18/07/16	20/07/16
Diseño del videojuego	20/07/16	15/09/16
Creación modelo 3D	20/07/16	23/07/16
Creación de texturas	22/07/16	23/07/16
Creación de escenas	25/07/16	6/09/16
Comportamiento de objetos	3/08/16	1/09/16
Pruebas de rendimiento	15/08/16	15/09/16
Pruebas en Unity	15/08/16	10/09/16
Pruebas en dispositivo	12/09/16	15/09/16
Memoria del proyecto	2/08/16	21/09/16
Búsqueda de documentación	2/08/16	12/08/16
Redacción de memoria	10/08/16	20/09/16
Validación de memoria	21/09/16	21/09/16

Figura 1. Estructura de tareas.

La representación de estas tareas, se realiza mediante el uso de un diagrama de Gantt especificando la relación que existe entre cada una de ellas.

El trabajo se divide en 3 bloques:

- Estudio previo: Aquí lo que se ha tratado es la adquisición de conocimientos previos para poder abordar el trabajo del TFG.
- Desarrollo del juego: Se desglosan los bloques más importantes del desarrollo del videojuego de este TFG.
- Memoria del proyecto: Consiste en la realización de esta memoria en la que además de explicar cómo se ha realizado el videojuego, búsqueda de información para dar consistencia al proyecto.

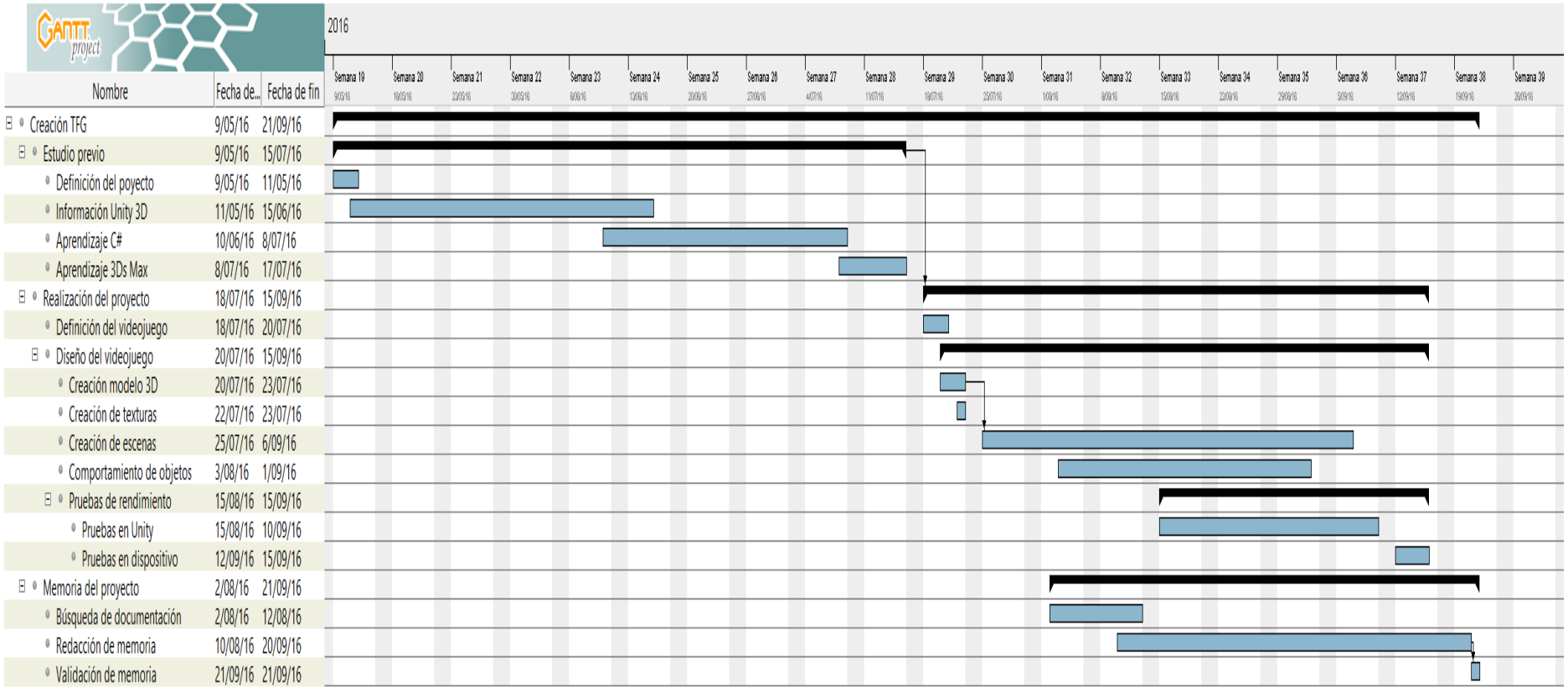

Figura 2.Diagrama de Gantt con la planificación temporal del proyecto.

1.4 Presupuesto

En este apartado se van a tratar los costes que ha tenido la realización de este proyecto, tanto el coste de personal como el material incluyendo la redacción de esta memoria. Se desglosarán en Recursos Humanos, Recursos Materiales y Costes Añadidos.

Autor:

Israel Matellán

Departamento:

Departamento de Informática.

Título:

"Desarrollo de una aplicación con Realidad Aumentada para dispositivos móviles"

Duración:

3 meses y medio.

RECURSOS HUMANOS

En esta sección aparecerá el coste de las personas participantes en el desarrollo del proyecto.

Persona	Categoría	Dedicación en meses	Coste Hombre/Mes	Coste €
David Griol	Ingeniero Senior	1	4.289,54 €	4.289,54 €
Israel Matellán	Ingeniero Junior	3.5	2.694,39 €	9.430,37 €
TOTAL				**13.719,91 €**

Tabla 1. Recursos humanos

RECURSOS MATERIALES

Aparecen todos los gastos materiales que se han necesitado durante todo el proceso.

Descripción	Coste €
Samsung Galaxy S7 Edge	799,00 €
Portatil Asus ROG GL552VW	849,00 €
Ordenador Sobremesa Mountain Steel	1.799,00 €
Disco Duro Externo WD 1 TB	59,00 €
TOTAL	**3.506,00 €**

Tabla 2. Recursos Materiales.

COSTES ADICIONALES

Se reflejan el resto de gastos ocasionados por el desarrollo.

Descripción	Compañía	Coste €
Fungibles		100 €
Licencia Windows 10	Microsoft	0 €
Unity 5	Unity Technologies	0 €
3Ds MAX 2017	Autodesk	0 €
Photoshop	Adobe	48,36 €
Transporte		50 €
Microsoft office 2013	Microsoft	0 €
ADSL	Vodafone	120 €
Vuforia	Qualcomm	0 €
TOTAL		318,36 €

Tabla 3. Costes adicionales

- **Fungibles**: Hacen referencia a material de oficina utilizado para el fin del proyecto.

- **Transporte**: Gastos de combustible para asistir a reuniones con el tutor.

Los precios de las licencias en su mayoría son gratuitos, debido a los acuerdos que tiene la universidad con las empresas desarrolladores del software.

TOTAL DE COSTES:

El total de costes según los apartados anteriores sería el siguiente:

Descripción	Coste €
Recursos Humanos	13.719,91 €
Recursos Materiales	3.506,00 €
Costes adicionales	318,36 €
TOTAL	**17.544,27 €**

Tabla 4. Total de costes.

CAPÍTULO 2: ESTADO DEL ARTE

2.1 Sistema Android

En esta sección se hará un recorrido a través del sistema operativo Android [2], los diferentes motores de videojuegos, la evolución de los videojuegos y la realidad aumentada.

El sistema operativo Android, es un sistema gratuito y libre basado en una variación del Kernel de Linux. Este sistema en la actualidad está bajo el dominio de Google.

Para poder desarrollar las aplicaciones de forma nativa en Android, Google pone a disposición de los desarrolladores un conjunto de herramientas denominado SDK en el que el lenguaje utilizado es JAVA.

Android a diferencia de otros sistemas operativos como es el caso de iOS (Apple) o Windows Phone (Microsoft) está basado en una estructura de desarrollo abierta pudiendo así los desarrolladores interactuar con las librerías que están dentro del sistema, pero esto no significa que se pueda instalar en los dispositivos la versión que uno quiera, puesto que son los fabricantes de los terminales quienes deciden cual es el que se puede instalar a no ser que se le realice un root al teléfono o Tablet, saltándose así estas restricciones. Haciendo esto, no se puede asegurar que las aplicaciones pueden funcionar correctamente.

La cuota de inserción de mercado es muy importante a la hora de pensar en desarrollar aplicaciones ya que eso puede significar el éxito o el fracaso de la misma. Según se puede ver en la siguiente gráfica, en el caso de España, el mercado de dispositivos móviles está copado prácticamente en su mayoría por dispositivos Android con un 92.8% de los dispositivos.

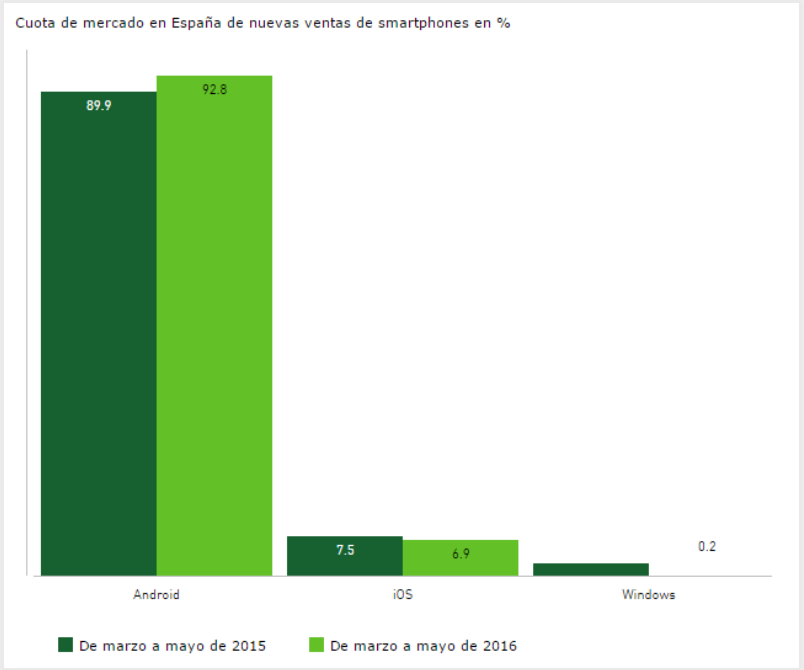

Figura 3. Cuota de mercado España

Tras ver esta gráfica no es de extrañar que la gente cree sus aplicaciones en esta plataforma ya que consiguen así llegar a más dispositivos con lo que más usuarios pueden hacer uso de ella.

Actualmente la gente de Mountain View (Sede de Google) han publicado hasta 14 versiones de su sistema todas ellas caracterizadas por tener el nombre de algún dulce o postre, yendo así desde la primera versión (v1.0) denominada “Apple Pie” hasta la última que se lanzó el pasado 15 de junio de 2016 (v7.0) “Nougat”.

Google ha hecho una gran apuesta con este sistema, ya que se han creado una infinidad de librerías desde el poder usar su sistema de mapas llamado "Google Maps" hasta el actual "Google Cloud Platform" donde se puede usar incluso la tecnología Big Data dando así una mayor capacidad de computo al sistema operativo.

2.2 Motores de videojuegos

Un motor de videojuegos o también denominado motor gráfico es un programa para poder crear y desarrollar videojuegos en diferentes plataformas como videoconsolas, ordenadores o dispositivos móviles.

Los motores de videojuego tienen que ofrecer al desarrollador que lo está usando las siguientes características:

- Motor de renderizado: Usado para poder pintar en pantalla lo que se está creando.
- Colisiones: Para poder detectar cuando un objeto ha podido interactuar con otro, actuando a dicha respuesta con la finalidad del videojuego.
- Entorno Multijugador: Necesario para poder generar juegos para varios jugadores a la vez.
- Sistema de Scripts: Sistema para poder dar al diseñador el control de los objetos de la escena mediante un lenguaje de programación. El lenguaje más usado en estos casos es C o sus derivados.
- Sonido: Haciendo que el usuario se quede inmerso en el videojuego.
- Inteligencia Artificial: Es una de las características más importantes dejando al lado el motor de renderizado ya que aumenta la vitalidad y experiencia del juego haciendo que en ocasiones pueda dar más dinamismo.

Hoy en día hay una gran cantidad de motores de videojuegos, ya que como se comentó anteriormente es una factoría la del videojuego muy lucrativo. Entre todos ellos destacan los siguientes:

- **Source 2 Engine [3]:**

Es el actual sucesor del aclamado Source Engine perteneciente a la empresa Valve. Con esta nueva versión, lo que se ha conseguido es una herramienta más potente para poder diseñar videojuegos. Se ha conseguido integrar además el uso de la Realidad Aumentada y la Realidad Virtual consiguiendo así que los desarrolladores puedan incorporar estas nuevas tecnologías a sus videojuegos. El lenguaje que se utiliza este motor es "C++". Permite la creación de videojuegos multiplataforma, ya no solo en PC, sino que también se pueden crear para dispositivos móviles con sistemas de iOS y Android. Juegos que se han desarrollado en este entorno son el aclamado "Half-Life", "Portal" o "Counter Strike" entre otros muchos.

- **CryENGINE [4]:**

Es un potente motor desarrollado por la empresa Crytek y usado por primera vez en el videojuego "Far Cry" para la plataforma PlayStation 4. Este motor puede usarse para crear videojuegos en diferentes plataformas como PC, PlayStation o XBOX. Destaca por la potencia que tiene a nivel de renderizado y físicas, animación y el poder iluminar las escenas en tiempo real, no necesitando en este caso, que se creen texturas de luz incorporadas en la escena. Es un motor de pago, con lo que muchos de los usuarios deciden no usarlo, además de que el aprendizaje de este motor es mucho mayor que el de otros como Unreal4 o Unity5.

- **Unreal Engine 4 [5]:**

Es el nuevo motor de la compañía Epic Game. Las características gráficas que aporta esta herramienta al igual que el uso de la iluminación que se le puede añadir son sus dos pilares fundamentales, haciendo más dinámico la creación de videojuegos en el entorno 3D. Tiene soporte multiplataforma, con lo que se pueden crear videojuegos para videoconsolas o dispositivos portátiles. El lenguaje de programación en los scripts de Unreal es C++. Algunos de los juegos que se han desarrollado con este motor son entre otros "Tom Clancy", "Batman: Arkham City", "Gears of War" o "Borderlands".

- **Unity 5.**

Unity 5 es la quinta versión del software de la compañía Unity Technology. Una de las principales características de Unity, es el poder exportar los proyectos con solo pulsar un botón a 21 plataformas distintas entre las que se encuentran las siguientes: Dispositivos con iOS, Android, Windows Phone, PC, Nintendo Wii, PlayStation, XBOX, Oculus Rift, Windows 10. Para poder trabajar con los scripts en esta plataforma existen 3 diferentes, C# (el más utilizado para el desarrollo de aplicaciones), JavaScript y Boo. Unity ofrece distintos tipos de licencias, desde la gratuita para poder desarrollar juegos de un nivel de requisitos no muy grande, hasta la versión PRO en la que todas las funcionalidades del motor a un precio de 125$ al mes.

2.3 Evolución de los videojuegos

Los videojuegos [6] se originan en la década de 1940 con la creación de los primeros superordenadores, empezando con el intento de creación de un juego de ajedrez surgido por la colaboración de Ala Turing con D.G. Champernowne. A principios de la década de los 60 se desarrolló "Spacewar!", que consistía en un duelo entre dos jugadores desarrollado en el MIT (Massachusetts Institute of Technology) ocupando en disco solamente 9K de memoria, pero no llegó a patentarse, ya que el hardware sobre el que estaba implementado era demasiado costoso como para llegar a los hogares de los usuarios, pero finalmente fue copiado por una de las empresas más fructíferas en el mundo del videojuego "Atari".

En la década de 1970 aparecieron las primeras máquinas recreativas basadas en el juego de "Spacewar!" creado en la universidad de Stanford. El mecanismo para poder jugar estaba controlado por una computadora "PDP-11" que se encargaba de manejar varias máquinas abaratando así los costes de fabricación, ya que estaba situado en los 20.000$ y las tiendas recreativas no podían costearse esos precios. Para que la gente pudiera jugar, tenía que disponer de monedas de 10 centavos que permitía al usuario jugar una partida. El gran hito de los juegos, llegó con la máquina recreativa "Pong" que es la versión comercializada del juego "Tennis for Two" de Higginbotham, creada por Al Alcom perteneciente a Atari. Fue presentado en 1972 y a raíz de ella, se implementaron grandes avances en este campo.

El país asiático Japón, apostó fuertemente en la década de los 80, creando la conocida empresa Nintendo, mientras que en occidente preferían otros modelos como el caso de Spectrum. En esta época, Estados Unidos sufría una gran crisis de videojuegos, pero consiguieron salir de ella, apostando por videoconsolas como la NES y el afamado juego "Tetris".

En 1985 salió al mercado un juego que causó un punto de inflexión, “Super Mario Bros”, ya que, en los juegos anteriores solo contaban con un número limitado de niveles, repitiéndose estos, pero en este caso, el juego tenía un fin no solo consiguiendo la máxima puntuación, sino llegar a salvar a una princesa de las manos de un enemigo que la tenía cautiva. Se trataba de un videojuego de plataformas en dos dimensiones, sirviendo a otras empresas para poder tener ideas brillantes para copiarlo.

Otra sección que creció considerablemente en los años 70, fue el de los videojuegos portátiles de la mano de la empresa californiana Mattel. El lanzamiento más importante de esta rama fue de Nintendo con la consola portátil “Game Boy”.

Durante la década de los 90, las videoconsolas dieron un gran salto por la introducción de las consolas de 16 bits y la llegada del CD-ROM como plataforma de almacenamiento de videojuegos. Con la llegada de ello, aparecieron en el mercado juegos sobre todo para PC en los que destacaba el uso de juegos 3D pre-renderizados como el caso de “Donkey Kong” y juegos de carreras que era un juego poligonal asentando así el uso de gráficos en 3D.

A finales de los 90, quien tuvo más presencia en el mercado fueron las consolas portátiles, saliendo a relucir una nueva versión de la Game Boy en la que los gráficos ya eran en color. Para la plataforma PC, los juegos que más éxito tenían eran los denominados FPS (First Person Shooter) que hacían que el usuario pudiera formar parte del juego ya que solo aparecía el arma que utiliza sin ver su avatar. Además, gracias a la conexión de internet, los usuarios podían jugar contra otros jugadores de forma remota apareciendo así los juegos de rol online (MMORPG).

Los principales competidores a principios del nuevo siglo serían Sony con su videoconsola PlayStation y Microsoft con XBOX, siguiendo hasta la actualidad copando el mercado de videoconsolas.

La plataforma con mayor número de usuarios es sin embargo el PC, debido a la gran potencia del hardware que disponen como tarjetas de gráficos más potentes pudiendo usar los juegos a una calidad extrema.

En la actualidad podemos clasificar los videojuegos según su género, es decir, según su mecánica. De esta forma, se tienen los siguientes tipos [7]:

- **Juegos Arcade**: Tienen una jugabilidad de acción rápida, típico de las máquinas recreativas sonde destacan los juegos de plataforma.
- **Juegos Multijugador**: Juegos en los que la principal característica es el poder jugar varios usuarios a la vez a través de la misma plataforma.
- **Juegos Online**: Jugados a través de internet indiferentemente de la plataforma usada para ello sin la necesidad de que los jugadores estén en la misma estancia.
- **Juegos MMO**: Combina las dos formas anteriores, es decir, son juegos online y multijugador, caracterizados por el uso de mapas y escenas para albergar a una gran cantidad de usuarios. Se tratan principalmente de juegos de estilo rol.
- **Juegos Sociales**: Juegos que utilizan las redes sociales o algún elemento de las mismas para poder desempeñar su finalidad. Son juegos en los que la plataforma principal es la web.
- **Juegos de navegador**: Son aquellos que se ejecutan a través de un navegador web o un plugin de los mismos por lo consiguiente son multiplataforma.
- **Juegos educativos**: Los más habituales son los de preguntas y respuestas o de aventura que como finalidad pretenden enseñar al usuario.

2.4 Realidad Aumentada

La realidad aumentada [8] es el término otorgado a la combinación del mundo real con elementos de la realidad virtual o elementos 3D, creando así una realidad mixta. Con ello se diferencia de la realidad virtual (RV) en la que todo ocurre fuera de visión real usando para ello el apoyo de unas gafas siendo todo creado por ordenador.

El término de realidad aumentada fue acuñado por Tom Caudell en 1992, pero con los avances actuales de la tecnología se ha hecho que las aplicaciones de realidad aumentada sean portátiles sin necesidad de tener potentes ordenadores para poder generarlo tal y como ocurría en esa época.

Para poder generar la realidad aumentada se necesita de una serie de elementos tales como:

- **Target**: Contenedor que hará de marcador para que el sistema reconozca donde tiene que realizarse la aparición del contenido virtual.
- **Cámara**: Será la encargada de reconocer el target activando así la realidad aumentada.
- **Unidad de proceso**: Encargada de procesar y gestionar los recursos para que la realidad aumentada funcione.

El tipo de realidad aumentada más utilizada en la actualidad, sobre todo en dispositivos móviles, es el denominado basado en marcadores. Este sistema consiste en el uso de un Target, generalmente una imagen o un código que servirá de marcador para poder trazar usando la cámara del dispositivo donde tendrá que aparecer los elementos 3D generados mediante software. Para realizar estos cálculos, el software tiene que interpretar el tamaño del marcador, y la distancia a la que se encuentra de la cámara, variando así los elementos que aparezcan.

Hay otros sistemas de realidad aumentada, como es la del posicionamiento de la persona mediante uso de la localización como es el caso del uso del GPS en los

dispositivos. Con ello lo que se consigue es no tener un Target físico al que asociar la realidad aumenta, sino que utiliza coordenadas de longitud y latitud para ello. Es un sistema útil para cuando se quiere tener una mayor flexibilidad y no depender de tener que escanear objetos. Un ejemplo de gran éxito de la realidad aumentada a través de la localización es el juego "Pokemon GO" de la empresa Niantic, ya que ha mezclado la realidad aumentada con un popular juego de finales de los 90.

Para conseguir la realidad aumentada se necesita de herramientas especializadas para ello. Una de las más potentes con reconocimiento de target es la que proporciona Vuforia [9], propiedad de la empresa Qualcomm. Vuforia proporciona un SDK, exportable a otras herramientas como el caso de Unity. Lo que hace es reconocer una escena y crear un frame con ello. Posteriormente, accede a una base de datos que el desarrollador ha creado con las imágenes que quiere reconocer y mediante software, hace que, sobre esa imagen creada, aparezcan los componentes de realidad aumentada que se necesiten.

CAPÍTULO 3:

DESARROLLO DE LA APLICACIÓN

Este Trabajo de Fin de Grado se va a centrar en el desarrollo de un videojuego en Realidad Aumentada para dispositivos móviles cuyo sistema operativo será Android. El desarrollo se hará con las siguientes herramientas:

- **Unity3D**: Será el motor de videojuegos utilizado debido a la cantidad de herramientas que nos proporciona para poder trabajar con la creación del videojuego.
- **Vuforia**: Plataforma para poder utilizar el reconocimiento de Targets y crear la Realidad Aumentada.
- **3D Studio Max**: Herramienta con la que se realizarán los modelos de ovnis usados.
- **Photoshop**: Software usado para poder crear las texturas de los ovnis y los Targets.

El videojuego consiste en destruir las diferentes naves espaciales que irán apareciendo cuando a través del dispositivo móvil reconozca el “Image Target” (a partir de ahora denominado “Marcador”). Una vez que estos aparezcan de forma progresiva,

pulsando la pantalla, saldrán unos rayos que los destruirán, haciendo que la puntuación del juego aumente, y en el caso de que los ovnis desaparezcan tras un cierto tiempo el usuario perderá vidas hasta finalizar así el juego.

3.1 Creación modelos 3D

Para crear los modelos que serán denominados “enemigos”, se utilizará el uso de las herramientas antes mencionadas 3D Studio Max [10] y Photoshop.

Lo primero será abrir el software 3D Studio Max, en este caso, la versión gratuita que tiene la universidad de Autodesk, 3ds Max 2017 [11], creando un nuevo proyecto, obteniendo así la siguiente interfaz:

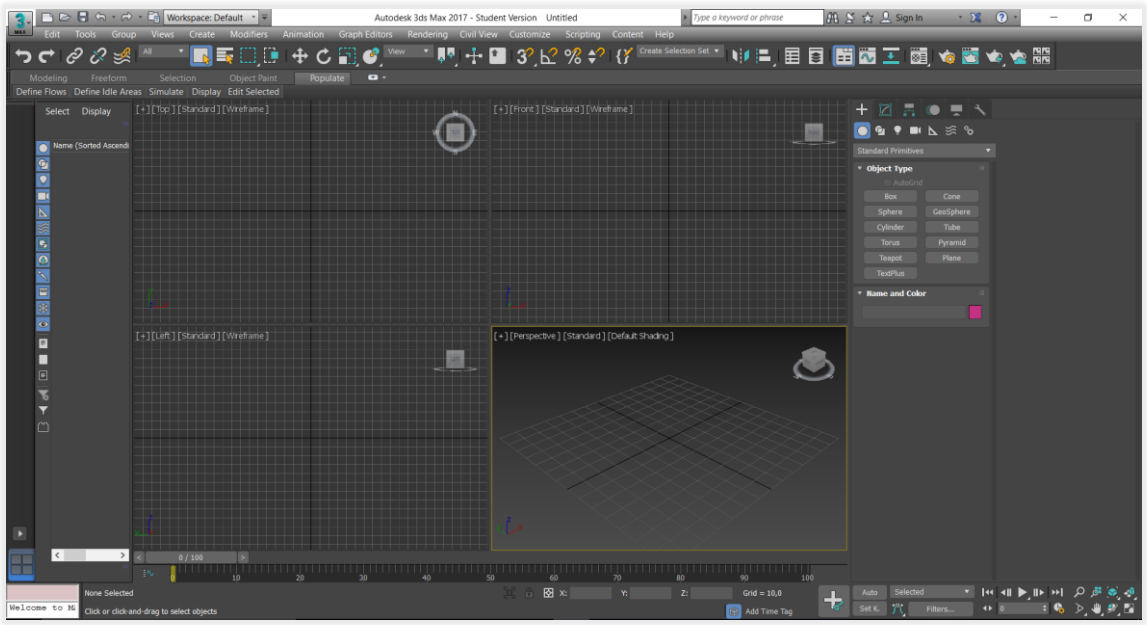

Figura 4. Proyecto nuevo 3ds Max

En primer lugar, para desarrollar el modelo que más tarde se usará en Unity para el videojuego será crear una primitiva estándar de tipo “Cilindro” que servirá como base para el desarrollo de nuestro objeto final.

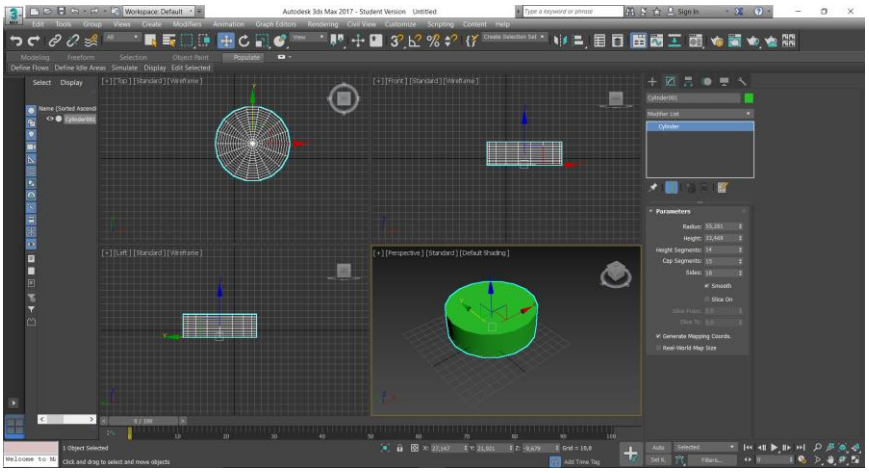

Figura 5. Creación de primitiva Cilindro

A continuación, se modificará la primitiva anterior para que pueda ser editada, así que se convertirá en un Editable Poly que permitirá poder manejar los vértices y caras de los que está compuesto el cilindro. Así, estirando las aristas, se obtendrá una forma más suave y ovalada como modelo. Adicionalmente para la creación de una protuberancia creando así más realismo se obstruirán y dilatarán las partes superior e inferior del mismo añadiendo herramientas de suavizado como los modificadores que tienen los elementos “soft”. Obteniendo así el resultado final de la malla.

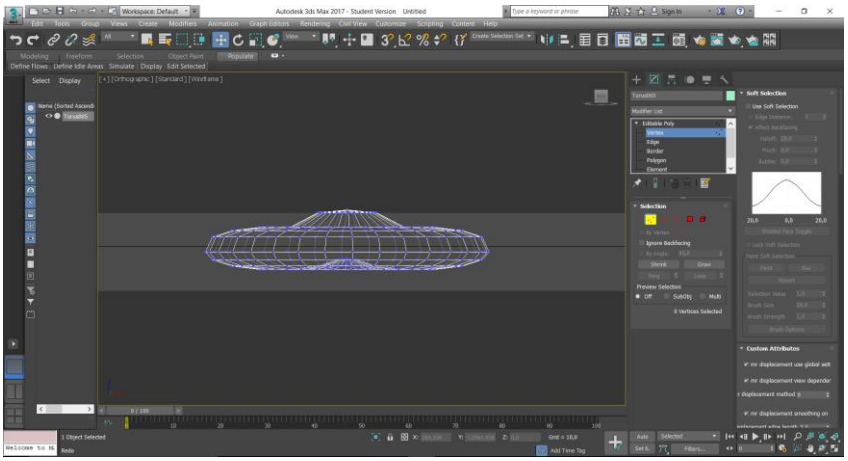

Figura 6. Modelo final sin textura.

Posteriormente se añadirá un material mediante la creación de una textura con la herramienta Photoshop [12]. Para el caso del ovni se crearán círculos concéntricos y diversas texturas que el propio Photoshop tiene para conseguir así un fichero.tar que se importará al editor de materiales de 3Ds Max. De esta forma se podrá trabajar también más tarde en Unity para el caso en que los materiales asociados a un objeto no se hayan importado correctamente, arrastrando el fichero a Unity se pueda reconfigurar.

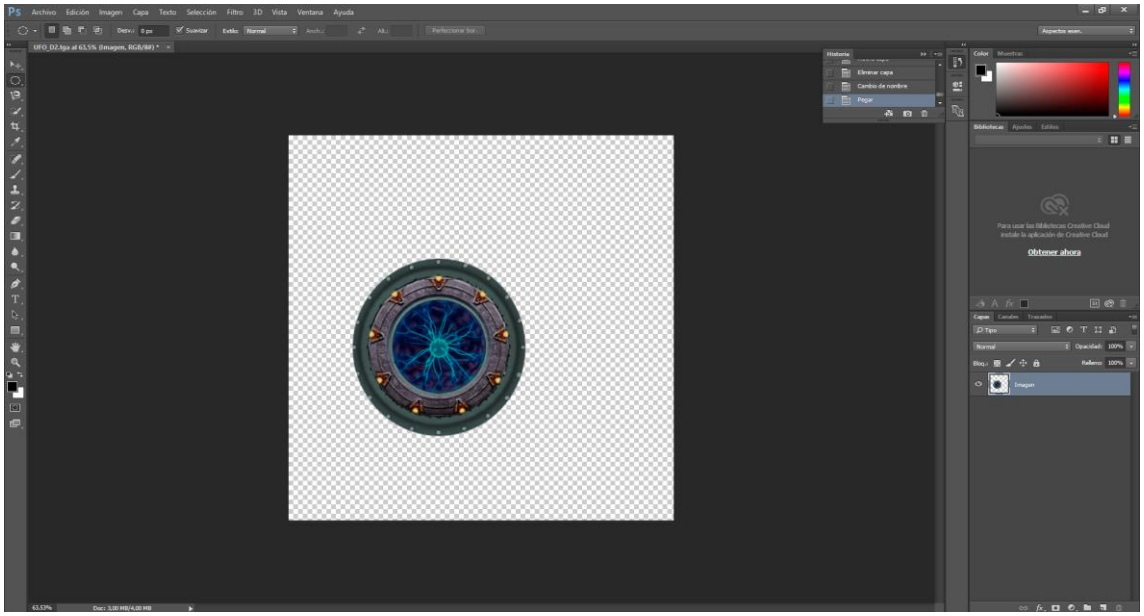

Figura 7. Textura Nave

Para poder usar el objeto creado desde la herramienta de diseño 3D se exportará mediante un fichero ".fbx" que es el formato que Unity mejor maneja para poder trabajar con modelos importados en su plataforma. Esto es así ya que se importa también las mallas por si se requiere modificar para temas de crear animaciones más tarde en Unity.

Figura 8. Modelo final OVNI

3.2 Creación del videojuego

Anteriormente se mencionó que el motor de videojuegos usado era el motor de Unity 3D, así que se procederá a registrar en la página del desarrollador y descargar desde la misma el instalador. Se encuentran muchas versiones, pero la utilizada en este proyecto será la versión Unity 5.4.0p1 (32-bit). Se usará el editor de 32 bits, ya que el

funcionamiento de Vuforia (a partir de la versión 6.0.117 sí que funciona) no soporta el editor de 64 bits y así se consiguen evitar problemas de compatibilidades.

3.2.1 Entorno de Unity

Una vez descargado e instalado Unity 3D [13], se procederá a la creación de un nuevo proyecto. En este caso al tratarse de la creación de un videojuego de Realidad Aumentada hay que usar un proyecto 3D obteniendo así un entorno como el siguiente.

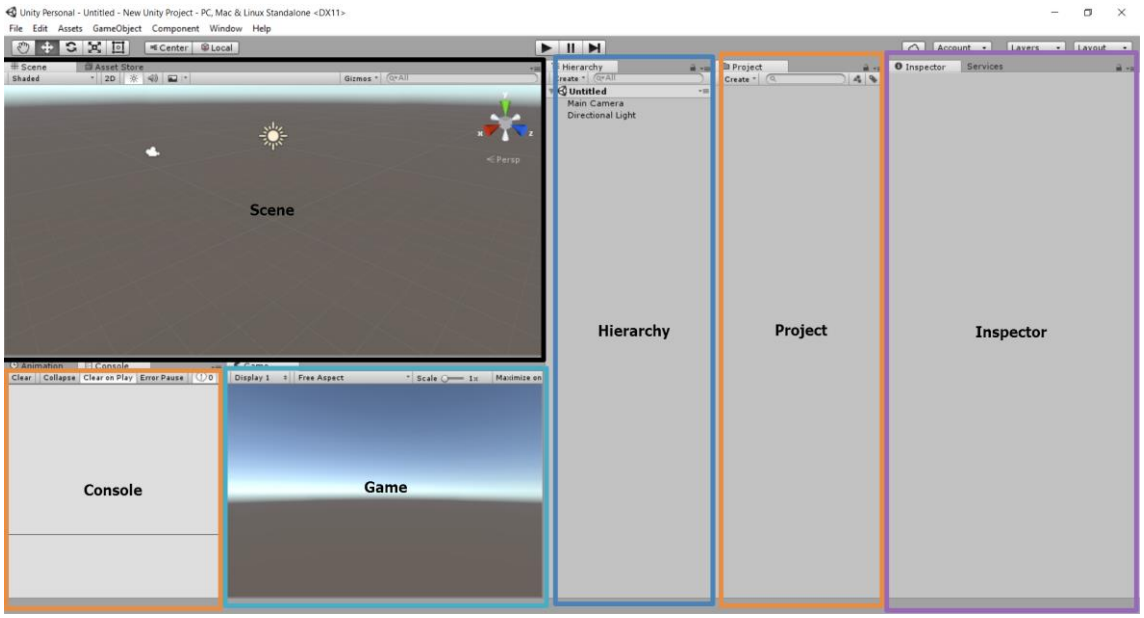

Figura 9. Pantalla Principal Unity

Unity está compuesto por diversas ventanas [14], todas ellas configurables, para hacer más fácil su manejo. Se comentan a continuación el uso de cada una de ellas:

- **Scene**: Es la ventana principal en la que aparecerán todos los objetos de nuestro videojuego de forma gráfica, pudiendo así modificarlos en el entorno 3D.
- **Hierarchy**: Aquí se crearán todos los objetos usados y las dependencias entre ellos: GameObjects, audios, cámaras…

- **Project**: Será el repositorio usado para almacenar cada uno de los elementos que se usarán en el proyecto en forma de carpetas.
- **Inspector**: Usado para poder ver las características y modificarlas de cada uno de los componentes creados.
- **Game**: Ventana de pre-visualización del juego para ver el funcionamiento del mismo en todo momento.
- **Console**: Consola de mensajes que irán apareciendo. Pueden ser mensajes del usuario para poder depurar el software, o mensajes del sistema a modo de Warning o Error.

Para poder abordar la ejecución del desarrollo del videojuego se darán una serie de nociones básicas, de las cuales las más importantes son las siguientes:

- **Creación de escenas**: Es el elemento principal de Unity, ya que las aplicaciones o juegos creados con Unity están basados en pantallas o escenas en este caso. Para ello basta con ir a la barra de menú y seguir la ruta File → New Scene. Las escenas se guardarán en la ruta indicada. Para que quede todo más ordenado se creará una carpeta en la jerarquía llamada "Escenas". En cada escena creada, aparecerá un cámara, la cual se puede modificar sus parámetros como la posición o rotación desde la ventana inspector o eliminar como en el caso de la escena principal del juego, ya que se utilizará la cámara de Vuforia para poder utilizar las funciones de la realidad aumentada.
- **Creación de objetos**: En Unity se pueden crear objetos de muchos tipos, como objetos 3D, objetos 2D, luces, audio, cámaras, elementos de interfaz de usuario, sistemas de partículas, u objetos vacíos desde la barra de menú en la pestaña GameObject. Los objetos vacíos serán útiles para poder crear funciones o añadir comportamientos necesarios en la aplicación sin necesidad de que haya que usar elementos con

componente física como los objetos 3D. Desde la ventana del inspector, se pueden modificar los atributos necesarios para lo que se desee.

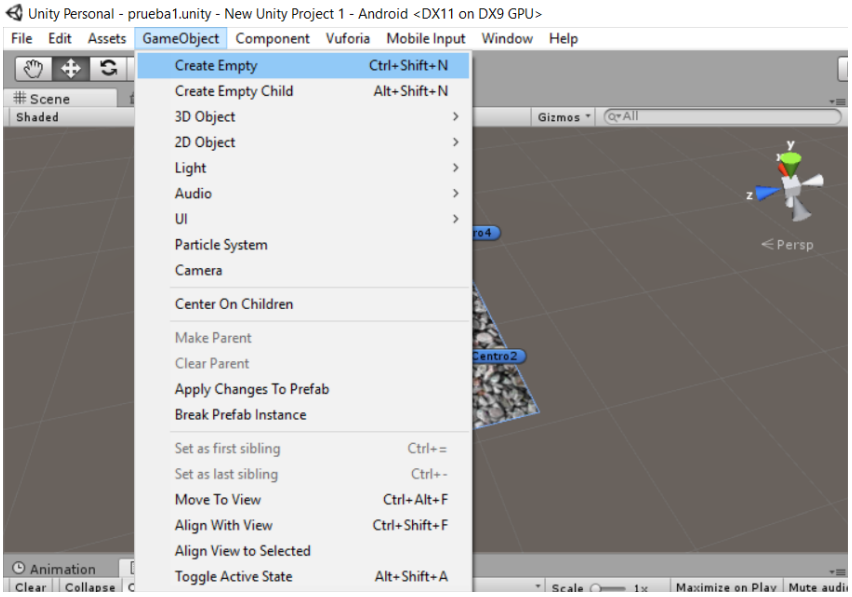

Figura 10. Creación de objetos.

- **Manipulación de objetos**: Los objetos creados en Unity, se pueden modificar de forma gráfica o mediante parámetros.

  Para manipularlos de forma gráfica, Unity tiene 5 herramientas básicas.

  -  Con ella el usuario puede moverse por la escena y visualizar como quedan situados los elementos.
  -  Se consigue trasladar los elementos que estén seleccionados en la escena en las coordenadas X, Y, Z.
  -  Utilizada para rotar sobre uno de los ejes los objetos según las necesidades.

-  Sirve para escalar los elementos que tengan componentes tridimensionales.
-  Al igual que la anterior, pero para elementos bidimensionales como textos o elementos de interfaz gráfica para el usuario.

Para manipular elementos mediante parámetros, se selecciona el elemento y a través del inspector se modifican la posición en la que aparecerá en la escena, la rotación que tendrá en la misma, o la escala o tamaño que tendrá ese objeto.

- **Concepto de Prefab**: Los prefabs son objetos de Unity, pero con la peculiaridad de que son objetos que el desarrollador puede reutilizar en cualquier escena del juego manteniendo las características con las que se creó. Las propiedades de estos objetos pueden modificarse al instanciarse en una escena sin que afecte al resto. Se identifican en los proyectos al tener un color azul por defecto. Si se desea modificar un objeto prefab, se arrastra este a la escena y se modifican sus parámetros en el inspector y una vez realizados los cambios, hay que aplicarlos dando click en el botón para ello en la misma venta. Si no se realiza esto, los cambios se han realizado en la instancia solo del objeto, pero no en el prefab del proyecto. Para crearlos, se puede hacer mediante dos mecanismos:
  - Crear un objeto Prefab en la ventana proyecto y una vez creado con el nombre requerido, arrastrar el objeto que se quiere tener como prefab, hasta el objeto prefab creado.
  - Arrastrar directamente desde la ventana jerarquía hasta la ventana proyecto.

3.2.2 Configuración del entorno

Se va a usar la realidad, así que lo primero es usar la plataforma de desarrollo Vuforia, y para ello previamente hay que crear una cuenta de desarrollador en su página principal [15].

Una vez que se ha realizado el registro, se creará una "License Key" que será utilizada para poder incorporarla en la cámara de realidad aumentada pudiendo así asociarla a un proyecto en el que tendremos los targets.

El siguiente paso, será tener una imagen que se usará de "Marcador" para que cuando el dispositivo la reconozca aparezcan los objetos 3D creados en Unity. Para ello en la pestaña de "Target Manager" se creará una nueva Base de Datos.

Figura 11. Imagen Target

La elección de esta imagen ha sido porque tiene diversas formas dentro de ella, que es lo que Vuforia utilizará para crear el seguimiento del target y mostrar así los objetos 3D en escena.

En la creación de un nuevo Target hay que indicar el tipo que se usará, el archivo donde se encuentra ubicado y las dimensiones de ancho (en el caso de una imagen simple) que tiene la imagen.

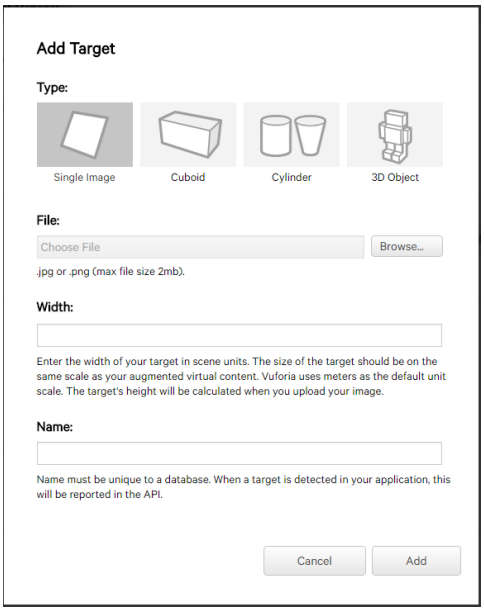

Figura 12. Añadir Target

Cuando ya esté añadida, aparecerán los diversos targets almacenados, así como la calidad y los puntos de trackeo que tiene la imagen.

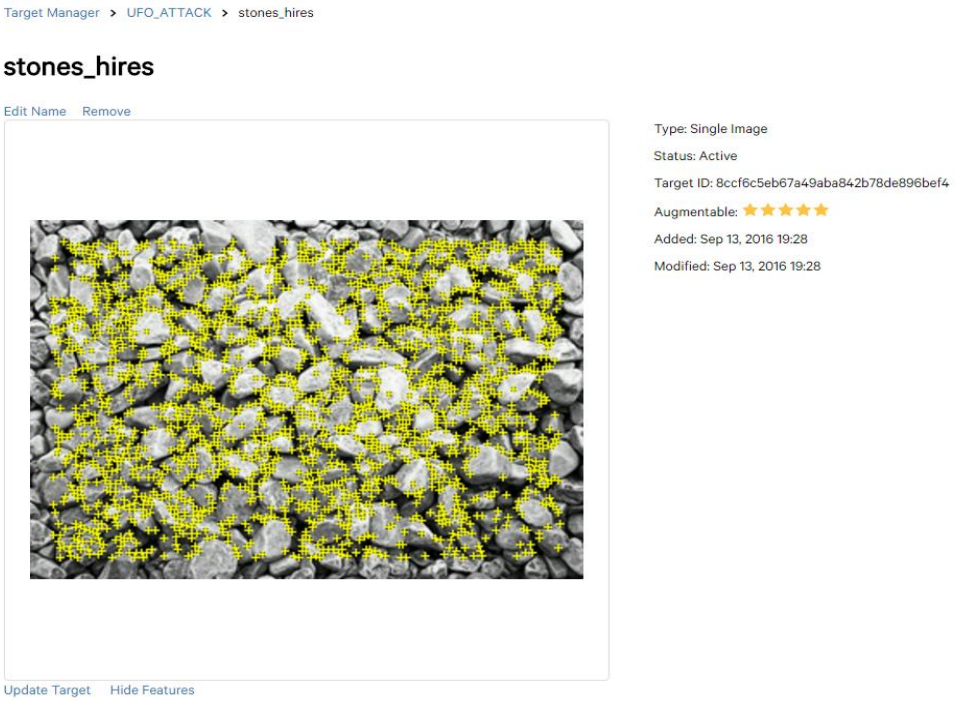

Figura 13. Image Target

Para poder integrarlo todo en el proyecto se descargará la base de datos que posteriormente se importará al editor de Unity, así como el SDK de Vuforia para poder tener los elementos necesarios para la creación de la realidad aumentada.

Para tener las cosas ordenadas en el proyecto, se utilizará la ordenación por carpetas, almacenando y organizando así todos los elementos que se necesiten durante el desarrollo del mismo.

Lo primero será importar los dos paquetes anteriormente descargados desde Vuforia mediante el uso de “Custom Package”.

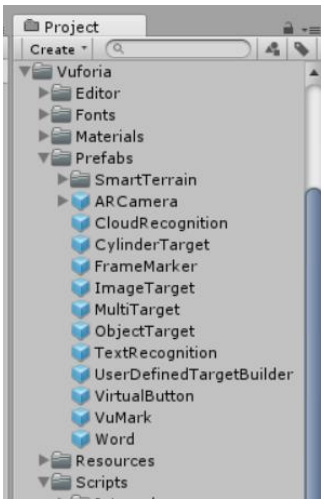

Figura 14. Carpeta Vuforia

Para crear la realidad aumentada en el videojuego hay que hacer uso de dos prefabs de Vuforia “AR Camera” e “Image Target”. Para usarlos se pueden arrastrar desde la ventana de Project a la escena o a la ventana Hierarchy.

La AR Camera aparte de ser una cámara la cual se usa para poder ver el juego cuando esté terminado, tiene la particularidad que creará los objetos tridimensionales cuando se reconozca el Target. Para poder conseguir esto, se tiene que añadir a la cámara la licencia creada en la página de Vuforia. Además, también hay que decir que tiene que cargar la base de datos con los targets.

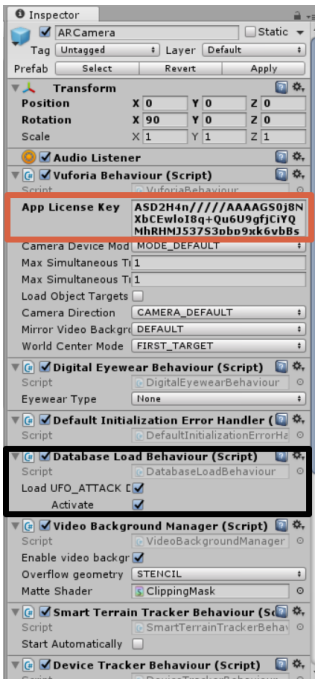

Figura 15. Características AR Camera

Por último, quedaría por configurar el prefab de Image Target, para ello lo arrastramos a la escena y nos aparecerá en la ventana de Hierarchy. Para su configuración hay que modificar en el Inspector el Script “Image Target Behaviour” que contiene el objeto, seleccionando que Base de datos y que imagen usar en el caso de tener varios.

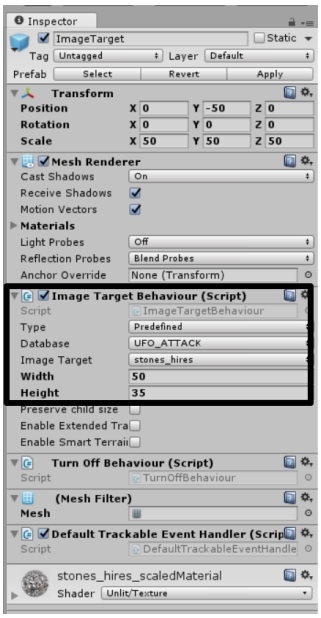

Figura 16. Características Image Target

Para crear una sensación de juego más completa se modificará uno de los scripts que contiene Vuforia que es el encargado de generar el trackeo de las imágenes. Lo que se quiere conseguir es que al perder de vista el marcador que se ha creado, es que el juego se pare momentáneamente hasta que se vuelva a reconocer haciendo que no avance el movimiento de los objetos. El script mencionado es "DefaultTrackableEventHandler.cs". En el script, bastará con añadir un elemento en el que obtendremos sus componentes para poder ejecutar un método en el cuál en el momento que se ha perdido el target pongamos el tiempo de ejecución del juego a 0 haciendo que no avance. Esto se consigue con:

```
Time.timeScale = 0f;
```

Además, el juego mostrará por pantalla mediante el uso de un texto que apunte al marcador para poder seguir adelante. Cuando el sistema haya reconocido de nuevo la imagen, el sistema volverá a avanzar, cambiando la sentencia anterior por:

```
Time.timeScale = 1f;
```

3.2.3 Generación de movimiento

Para insertar un modelo 3D, lo primero es importar el material que utiliza el objeto 3D para que así una vez importado el modelo, el sistema enlaza automáticamente la textura con el modelo. De no hacerlo así, habría que enlazarlo de forma manual para poder usarlo más tarde.

Se puede hacer mediante dos formas:

- Arrastrando desde el directorio local del ordenador en una carpeta del Inspector.
- Haciendo click derecho y seleccionando "Import New Asset" y eligiendo el archivo que se quiere importar.

Para poder ver cómo queda el objeto, hay que arrastrarlo a la escena. Se puede visualizar mediante la ventana Game, pero el objeto se verá con poca luz. Esto se debe a que no se ha agregado ningún foco de luz en la escena. En Unity se disponen de varios tipos de luces los cuales se añadirán mediante la barra de herramientas GameObject → Light → Directional Light.

Para crear más realismo al videojuego, se usará la propiedad que tiene Unity de jerarquía, en la que se puede crear dependencias padre e hijo, así que, arrastrando el objeto creado de luz dentro de la cámara se consigue que haya donde vaya la cámara el punto de luz irá con ella.

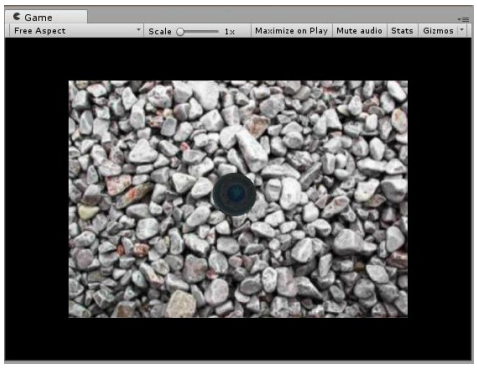

Figura 17. Escena sin luz.

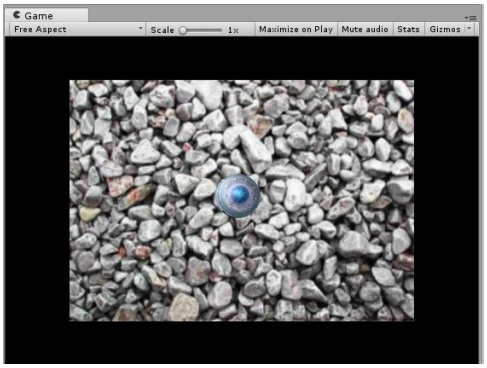

Figura 18. Escena con luz

El movimiento estará basado en el movimiento del ovni alrededor de un punto que previamente creado en el image target y que se irá alejando paulatinamente de ese punto aumentando además el radio de giro.

Lo primero será crear un objeto vacío (GameObject Empty) en las coordenadas (0,0,0). Para crear el movimiento generaremos un script C# que llamaremos “Rotar.cs” que se asignará al objeto ovni desde el inspector. Unity tiene su propio editor de código “MonoDeveloper” que ayudará para crearlos [16].

```
using UnityEngine;
using System.Collections;

public class Rotar : MonoBehaviour {

    public Transform objetoCentroDeRotacion;
    public float rotacionPorSegundo = 45f;
    public float radioInicio = 0f;
    public float incrementoRadio = 0.5f;

    void Update ()
    {
        radioInicio += incrementoRadio * Time.deltaTime;
           transform.position = new Vector3 (objetoCentroDeRotacion.position.x, transform.position.y, objetoCentroDeRotacio
n.position.z);
        transform.Translate (-radioInicio, 0, 0);
        transform.RotateAround (objetoCentroDeRotacion.position, Vector3.up, rotacionPorSegundo * Time.deltaTime);
    }
}
```

Este es el script Rotar. Por defecto los scripts creados en Unity tienen dos funciones principales (no es obligatorio su uso):

- **Start ():** Será usado para que se inicialicen las constantes o funciones necesarias al empezar a usar el script.
- **Update ():** Se ejecutará periódicamente por cada frame del juego. Muy útil para poder realizar ejecuciones periódicas como para realizar movimientos de objetos.

Desde un script se pueden modificar objetos de Unity instanciándolos al principio del script. Para la generación del movimiento es necesario crear varias variables que ayudarán para crear el movimiento (rotacionPorSegundo, radioInicio, incrementoRadio) y un objeto tipo Transform que será el que pase por parámetros la posición sobre la que girará el ovni. Para ello en la función Update () se irá modificando la posición y la rotación del ovni mediante las sentencias:

- `transform.position = new Vector3() // Vector para poder trabajar con las coordenadas.`
- `transform.Translate() // Moverá a la posición en la que se encuentra el punto desde donde girará.`
- `transform.RotateAround() // Función predefinida para poder girar.`

En la función "RotateAround()" hay que indicar la velocidad con la que girará mediante el uso de "Time.deltaTime" para que el juego funcione de forma independiente al frame rate, ya que si no dependerá de la velocidad del procesador de nuestro dispositivo con lo que se verá de forma diferente de donde lo ejecutemos.

Una vez creado, se procede a crear un prefab, que se utilizará para poder generar varios enemigos más adelante. Para ello, basta con arrastrar el objeto que hemos creado a la ventana Project y se obtendrá un Prefab pudiendo así instanciarlo desde código todas las veces necesarias.

3.2.4 Creación de disparos

Para poder seguir con la funcionalidad del juego, hay que crear un elemento disparo. El disparo será un elemento Prefab, al igual que el caso del ovni que será llamado cada vez que el usuario toque la pantalla [17].

Unity tiene una tienda en la que se pueden descargar diversos elementos creados por terceras personas. En este proyecto se usará un generador de partículas que simulará el efecto de un disparo láser. En este caso se usará un paquete de partículas de la Asset Store de forma gratuita importándolo al proyecto.

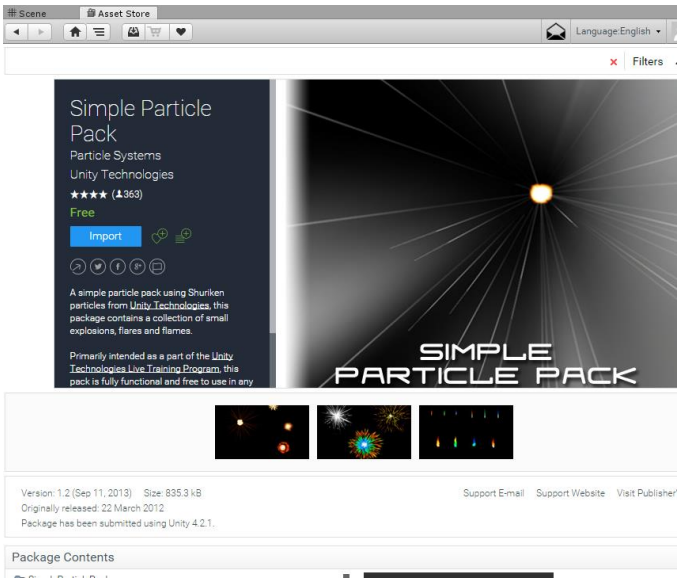

Figura 19. Paquete Partículas Asset Store

Una vez importado al proyecto, bastará con seleccionar el tipo de partículas que formará parte del disparo y arrastrarlo a la escena. Para poder trabajar con ello es necesario crear un GameObject vacío para poder crear un Prefab del disparo y añadirle el componente de partícula que se ha generado. Se necesita que el objeto tenga un mecanismo "Rigibody" que será usado para poder manejar el movimiento del disparo.

El comportamiento del disparo se generará mediante un script denominado "Disparo.cs".

```
using UnityEngine;
using System.Collections;

public class Disparo : MonoBehaviour
{
    private Rigidbody rb;

    void Start () {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate(){
        rb.AddForce (transform.forward * 300);
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.name == "Suelo") {
            EliminarDisparo ();
        } else if (other.tag == "Enemigo") {
            EliminarDisparo ();
            other.SendMessage ("Muere", SendMessageOptions.DontRequireReceiver);
        }
    }

    void EliminarDisparo (){
        Destroy (gameObject, 0.5f);
    }
}
```

Este script conseguirá que el objeto se mueva añadiendo el método "`AddForce()`" que proporcionará a un movimiento en un eje continuo al componente rigibody. Adicionalmente se le añaden varios métodos:

- `void OnTriggerEnter (Collider other):` Usado para detectar si el disparo ha colisionado con algún elemento de la escena.
    - Si el objeto ha colisionado con el image Target denominado "Suelo" el disparo se destruirá usando para ello el método EliminarDisparo ().

  - Si el objeto disparo ha colisionado con un enemigo (que se detecta con el uso de los tags) se destruirá el disparo y mandará un mensaje al sistema con el uso de SendMessage() para que como se verá más adelante se destruya el objeto ovni.
- `void EliminarDisparo ():` Este función llamará al método de Unity Destroy () que conseguirá que desaparezca de la escena el objeto que metamos en su interior.

Conseguido este script, se le incorporará al prefab Disparo. Ahora lo que hace falta es generar los disparos de forma periódica con lo que es necesario crear un GameObject que se pondrá dentro de la cámara para que en cuanto el usuario lo necesite instancie los disparos necesarios. Para ello se crea un script denominado "Disparar.cs" que tendrá la siguiente estructura.

```
using UnityEngine;
using System.Collections;

public class Disparar : MonoBehaviour {

    public Transform disparoPrefab1;
    public float intervaloDisparos = 1f;
    private float siguienteDisparo = 1f;

    void Update () {
        if (Input.touchCount > 0 || Input.GetButtonDown ("Fire1") && Time.time > siguienteDisparo) {
            siguienteDisparo = Time.time + intervaloDisparos;
            Instantiate (disparoPrefab1, transform.position, transform.rotation);
        }
    }
}
```

Lo más importante de esta clase, es que se utiliza el método "Instantiate ()" que es usado para poder generar de forma automática en la escena los disparos que saldrán

referenciados con la posición de la cámara. Para controlar la cantidad de disparos que se generan, se utiliza la condición de que se haya tocado la pantalla y que haya pasado un cierto tiempo entre ellos. En este caso hay que utilizar el tiempo actual del tiempo y compararlo con el tiempo deseado entre los disparos.

3.2.5 Generación y destrucción

Para que dos objetos puedan colisionar en el espacio generado en Unity lo primero es que las coordenadas de cada uno de ellos tienen que coincidir, superponiendo así ambos objetos. Además de eso, estos objetos tienen que tener componentes colliders. Existen varios tipos dependiendo de las necesidades de cada uno.

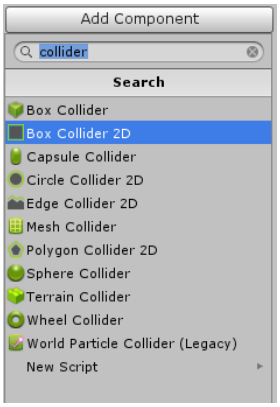

Figura 20. Tipos de Colliders

Como se quiere que al colisionar un disparo con un ovni (enemigo) este último se destruya, hay que añadir al prefab creado del ovni uno de los colliders anteriores. Normalmente se asignan dependiendo de la forma que tenga el objeto para así tener una sensación de juego más real, sobre todo en caso como en este videojuego de shooter.

En las propiedades de los colliders hay una que se denomina “Is Trigger” que se utiliza para que el componente que lo tenga activado devuelva al sistema la información sobre qué es lo que ha colisionado con dicho objeto, pudiendo así reaccionar ante tal

mensaje. En este caso se activará el componente en el Disparo para que informe al sistema de la colisión con el ovni y ejecutar el comando Destroy () para eliminarlo de la escena, aumentando así la puntuación del usuario.

La generación de forma periódica de los enemigos se realiza mediante el anteriormente usado método Instantiate () para mediante una posición y un periodo de tiempo que tenga que transcurrir se generen los ovnis. En el caso de este juego, se colocarán cinco puntos repartidos por el image target y a distintas alturas para aumentar el nivel de dificultad. Al principio se generarán desde un punto, pero a alcanzar cierto tiempo jugando, empezarán a aparecer los ovnis desde otras posiciones.

Cuando llegan a una cierta distancia los ovnis estos desaparecerán haciendo que el jugador pierda vidas y en el momento que las vidas se agoten, el jugador perderá el juego.

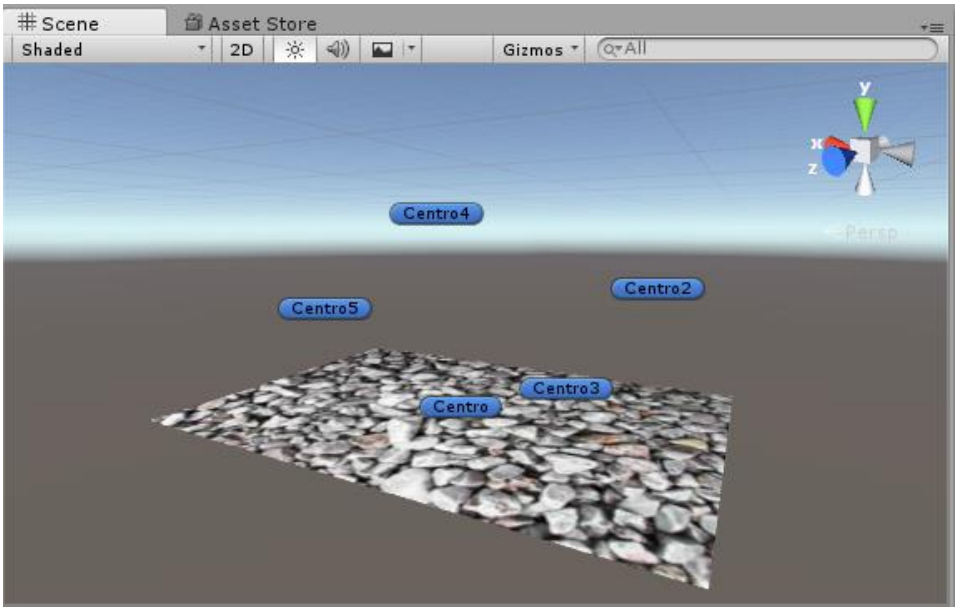

Figura 21. Puntos de generación de ovnis

3.2.6 Puntuación y vidas

La finalidad del juego es conseguir la mayor puntuación posible antes de que las vidas lleguen a 0. Lo primero para poder hacer la puntuación y las vidas es crear una UI (de las siglas en ingles de Interfaz de Usuario) y Unity para ello provee de una herramienta denominada canvas que es el entorno al que el usuario tiene acceso. En ella se pueden crear textos, botones, imágenes, campos de texto…

El canvas tendrá una imagen para almacenar las vidas y un texto para la puntuación. Para ello hay que crear una serie de imágenes que se almacenarán en un array que se irá recorriendo a medida que el usuario pierda vidas al escaparse los ovnis, creando así las vidas del usuario.

Ilustración 1. Escena Principal.

Para la puntuación se irá creando un contador para actualizándolo periódicamente al destruir un ovni. La cantidad de puntos a añadir al contador dependerá de la distancia a la que el enemigo se encuentre, siendo mayor esta cuanto más alejado esté ya que la velocidad con la que gira el ovni es mayor. La puntuación

obtenida finalmente por el usuario, será almacenada para detectar si ha sido la máxima puntuación obtenida en el dispositivo o no mediante el uso de las funciones internas de Unity para esta finalidad denominado “PlayerPrefs()”. En el caso de que se quiera almacenar un cierto valor se usa el método que contiene PlayerPrefs.SetInt() y para poder recuperarla se utiliza PlayerPrefs.GetInt(). En el caso de querer recuperar un valor, es preferible añadir un valor por defecto para las ocasiones en las que no se tenga todavía un valor previo ya que sino devolverá un error de datos. Siempre se utiliza una palabra denominada “token” que servirá para poder acceder a los valores que se necesiten, teniendo así la posibilidad de guardar todos los datos necesarios. Para el caso de la puntuación máxima que se tenga durante todas las partidas, se usará la palabra “highscore” y para almacenar el valor de la puntuación que el jugador tiene en la partida actual, se usará “actual”. Estos valores estarán siempre disponibles desde cualquier escena ya que el tipo que se le asignan a las variables anteriores son del tipo internal. Este tipo de variables, son como las privadas, pero con la particularidad de acceder a ella no solo desde el lugar donde se declararon. Todo esto se desarrolla en el script “EstadoJuego.cs”.

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class EstadoJuego : MonoBehaviour {

    public Sprite[] vidasImagenes;
    public int vidasActuales = 0;
    public int vidasIniciales = 3;
    public Image guiVidas;
    public  int puntuacion = 0;
    public Text guiPuntuacion;

    internal bool gameOver = false;
    internal int highscore = 0;
    internal int actual = 0;

    void Start () {
        highscore = PlayerPrefs.GetInt ("highscore", 0);
        actual = PlayerPrefs.GetInt ("actual", 0);
        vidasActuales = vidasIniciales;
        guiVidas.GetComponent<Image> ().sprite = vidasImagenes[vidasActuales];
        puntuacion = 0;
        ActualizarPuntuacion ();
    }

    public void PerderVida()
    {
        if (vidasActuales > 0)
        {
            vidasActuales--;

        }
        if (vidasActuales < vidasImagenes.Length)
        {
            guiVidas.GetComponent<Image> ().sprite = vidasImagenes[vidasActuales];
        }
        if (vidasActuales <= 0)
        {
            SendMessage ("PartidaTerminada", SendMessageOptions.DontRequireReceiver);
        }
    }

    public void IncrementarPuntuacion (int valorAIncrementar)
    {
        puntuacion += valorAIncrementar;
        ActualizarPuntuacion ();
    }

    public void ActualizarPuntuacion()
    {
        guiPuntuacion.GetComponent<Text> ().text = puntuacion.ToString ("D5");
    }
}
```

3.2.7 Creación de escenas adicionales

La pantalla inicial del juego no será la que contenga la cámara de realidad aumentada. Para ello se crea una pantalla inicial, la cual servirá de presentación y habrá dos posibilidades mediante botones:

- Botón “Empezar”: Hará que se pase a la pantalla principal en la que se activará la realidad aumentada en el que el usuario enfoque al marcador comenzando así el juego.
- Botón “Instrucciones”: Mostrará las instrucciones del juego. Adicionalmente tendrá un botón que enlazará con un repositorio para poder descargar el marcador que el usuario tendrá que utilizar para poder empezar a jugar.

Figura 22. Escena portada.

Cuando el usuario haya terminado de jugar, es decir, no tenga más vidas, la aplicación cargará una nueva pantalla que estará formada por texto en el que se le mostrará al usuario si su puntuación es nuevo record o no dentro de su dispositivo móvil.

También contendrá un botón que llevará al usuario a poder volver a jugar llevándole a la pantalla de portada anteriormente mencionada.

Figura 23. Escena GameOver.

Para la pantalla de instrucciones, se indicará mediante el uso de un elemento texto del canvas, las instrucciones para poder descargar el image target que será necesario para poder jugar. Esta imagen estará alojada en un repositorio online. Adicionalmente, se podrá mediante el uso de dos botones, volver a la pantalla anterior denominada “Portada” o empezar directamente a jugar.

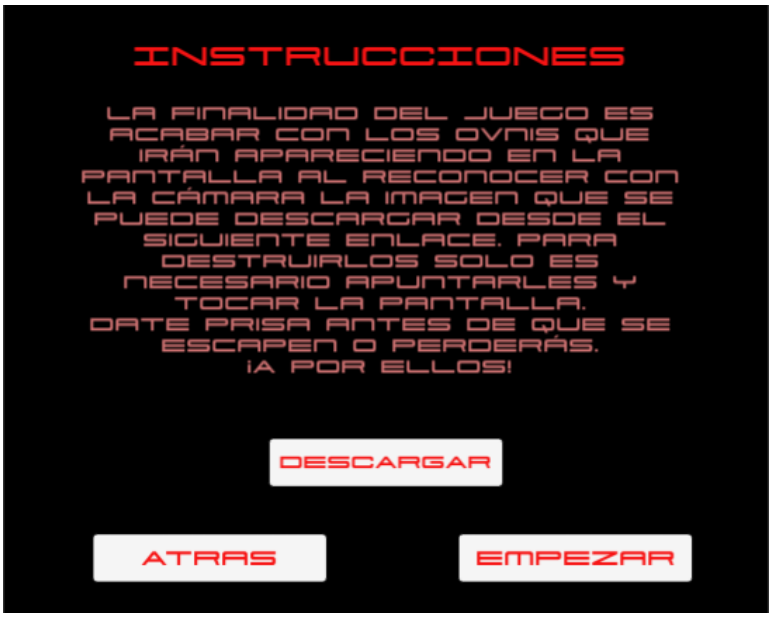

Figura 24. Escena Instrucciones.

Estas pantallas están compuestas básicamente por UI. El principal problema es el de los diversos tamaños de pantalla que tiene los dispositivos Android. Para paliar esto, Unity tiene la posibilidad de que sus canvas se redimensionen dependiendo de la cantidad de pixeles que tenga. Además, los elementos del canvas se pueden anclar a una posición del mismo tomándolo así de referencia mediante lo que se denomina "Anchor" para guardar así las dimensiones que se le establecen en el inspector.

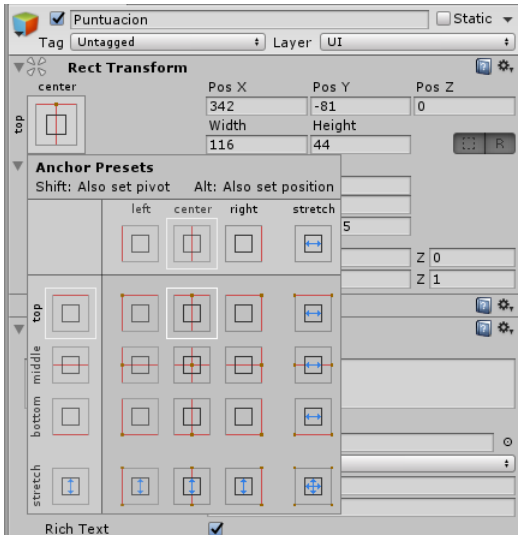

Figura 25. Anchor

Para la navegación entre pantallas se utiliza la función `SceneManager.LoadScene()`. Recibirá por parámetro el nombre de la escena a la que se pasará. En el caso de la portada dependiendo del botón presionado se irá a la principal con el juego o a la de instrucciones. En el caso de la principal la siguiente pantalla será la de GameOver y a través de ella al pulsar el botón de reinicio se irá a la portada principal para poder volver a jugar otra partida.

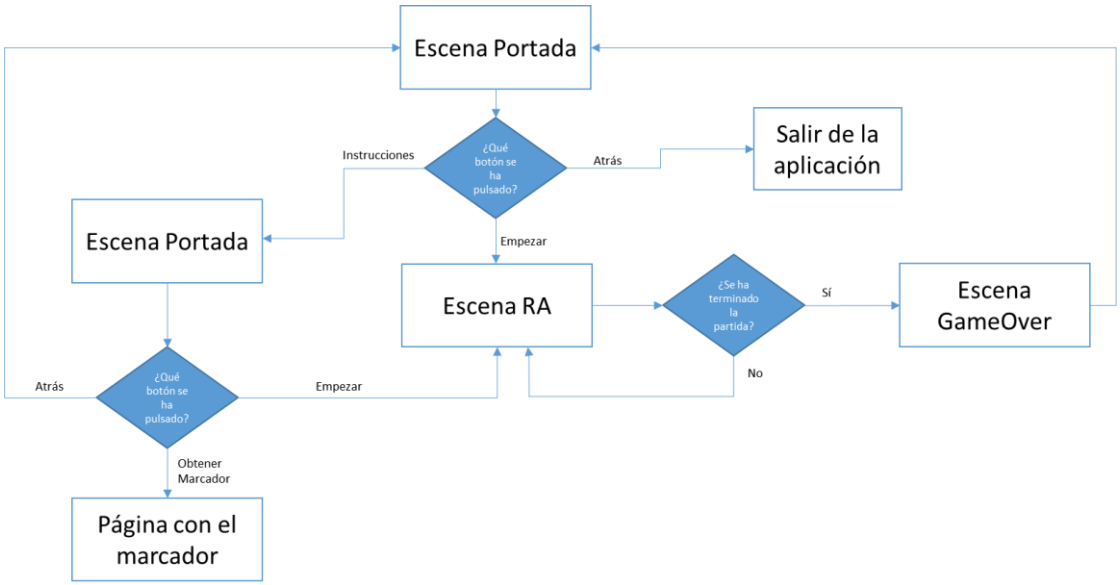

Figura 26. Flujo de la aplicación

También se puede navegar con el botón físico del dispositivo que hará que en el caso de que el usuario esté en alguna de las pantallas que no sea la portada vuelva a la anterior siguiendo el flujo anterior, pero en el caso de estar en la portada, el usuario saldrá de la aplicación. Esto se implementa en el código mediante el uso de un switch.

```
void Update () {
        if (Input.GetKeyDown (KeyCode.Escape))
        {
            switch (accion) {
            case Acciones.Salir:
                Application.Quit();
                break;
            case Acciones.CargarEscena:
                SceneManager.LoadScene (nombreEscena);
                break;
            }
        }
    }
```

3.2.8 Efectos de sonido

Cualquier aplicación que se precie, necesita tener música y efectos de sonido. Para poder implementarlo hay que diferencia de lo que son los efectos de sonido, por ejemplo, cuando un rayo colisiona contra un enemigo de lo que es la música de fondo. En los efectos, se añadirá al Prefab Ovni y Disparo, un Audio Source. Un Audio Source es un contenedor en el que se le incluye la pista de sonido que se necesite y en el caso de que se quiera cuando se genere, bastará con marcar la opción “Play on Awake”.

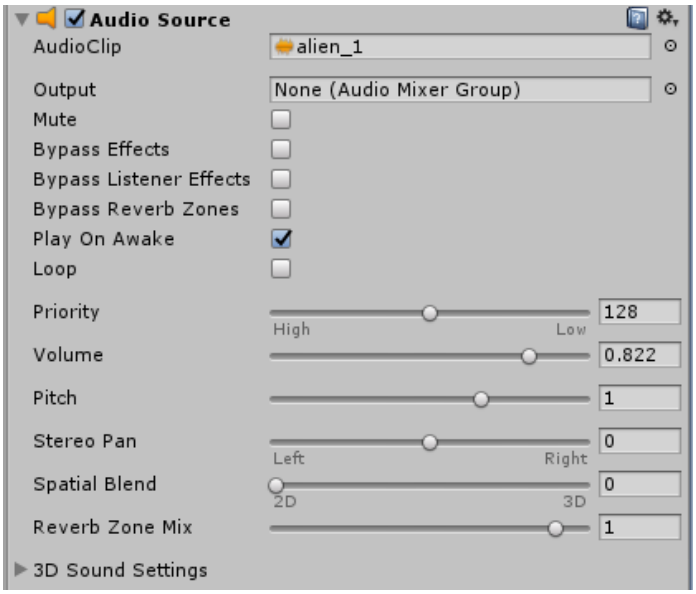

Figura 27. Audio Source

Para la música de fondo, el sistema cambia un poco, ya que se necesita que esté en todas las escenas. El problema es que Unity al cambiar de una escena a otra, elimina todos los elementos que se hayan generado, así que hay que crear un elemento que contenga además del Audio Source, un script en C# [18] como el siguiente:

```
using UnityEngine;
using System.Collections;

public class NoDestruir : MonoBehaviour {

    void Awake()
    {
        DontDestroyOnLoad (transform.gameObject);
    }
}
```

Con este script lo que se consigue al llamar a la función DontDestroyOnLoad (), es que el objeto que le pasemos no sea destruido al pasar de escenas, con lo que siempre estará presente.

3.2.9 Exportación de proyectos

A la hora de terminar el proyecto, hay que conseguir un ".apk" y para ello se utiliza "Build Settings…" de la barra de menú.

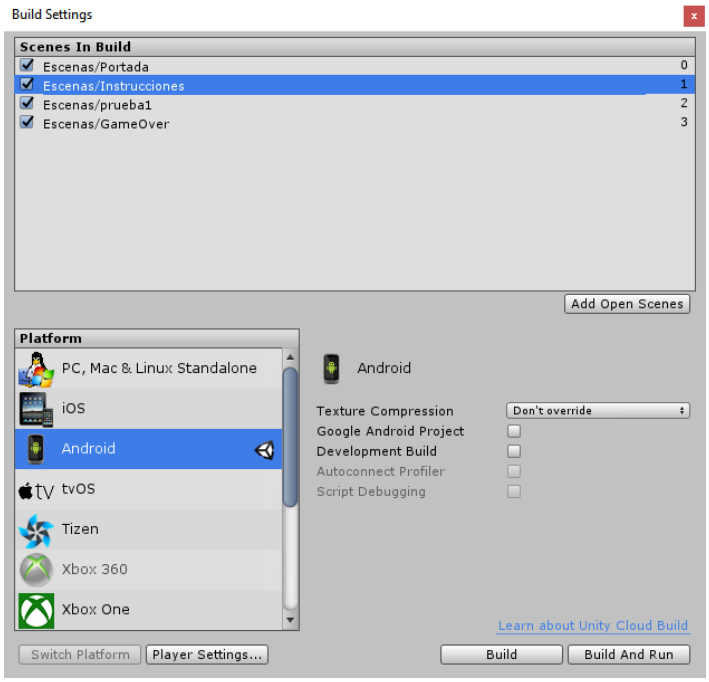

Figura 28. Build Settings.

Para este proyecto, como se creará una aplicación Android se seleccionará y modificarán los "Player Settings". Esto último es muy importante ya que se indicará el nivel mínimo de API que necesitará el dispositivo móvil que tiene que tener para poder instalar la aplicación, la orientación que puede tener la aplicación y el icono que aparecerá al crearlo.

Además para que Unity deje crear el ".apk" hay que definir un nombre de compañía y un nombre de producto que se añadirá a la identificación del producto final.

Una vez configurado correctamente solo hará falta pulsar el botón "Build" y definir la ruta donde aparecerá la aplicación para instalarla en el dispositivo móvil.

CAPÍTULO 4

CONCLUSIONES Y TRABAJO FUTURO

4.1 Conclusiones

Una vez terminado este trabajo de fin de grado y viendo todo lo que se ha hablado, se puede concluir que el uso de las tecnologías hace que los videojuegos avancen a unas nuevas fronteras para entretener al usuario.

El uso de nuevas plataformas de nuevos motores gráficos facilita enormemente el desarrollo de videojuegos como ocurre con Unity 5. El uso de esta herramienta que está muy orientado a la ejecución gráfica consigue que gente sin mucho conocimiento de programación puedan hacer nuevo software de entrenamiento para diversas plataformas solo haciendo click a un botón.

El uso de la realidad aumentada es una tecnología de moda haciendo que los videojuegos sigan la senda de compaginar los entornos reales que la persona tiene alrededor con el uso de componentes virtuales haciendo más satisfactoria la experiencia del juego.

Este proyecto ha conseguido dar una pequeña visión de lo que puede ofrecer al desarrollo de videojuegos el uso de la realidad aumentada en dispositivos móviles.

4.2 Trabajo futuro

Tras la realización del proyecto se ha visto que hay varios caminos para poder seguir investigando con la Realidad Aumentada y los videojuegos:

- Realizar versión multijugador, en la que junto a otras personas desde sus dispositivos móviles puedan jugar con un mismo target.
- Adaptación del videojuego a otras plataformas, tanto móviles como para videoconsolas.
- Mejorar la inteligencia artificial de los enemigos haciendo que aparezcan en otros lugares solo sabiendo donde no está apuntando el jugador para poder aumentar así la dificultad de los niveles.
- Uso de la geolocalización para no tener que depender de targets físicos e interactuar de esta forma más con el entorno que rodea al usuario.

Glosario

Unity	Motor de videojuego multiplataforma creado por Unity Technologies.
Android	Sistema operativo multidispositivo libre desarrollado por Google.
Realidad Aumentada	Término para describir a la mezcla de la realidad con elementos digitales 3D.
Image Target	Representación de imagen usada en la realidad aumentada para localizar el lugar donde aparecerán los elementos tridimensionales.
Prefab	Objetos reutilizables en Unity.
Script	Archivo de programación para poder dar instrucciones.
Videojuego	Juego electrónico que se visualiza mediante una pantalla.
Motor de Videojuegos	Conjunto de rutinas que permiten el diseño y creación de videojuegos.
Escena	Contenedor de objetos de Unity para poder crear diversas pantallas que verá finalmente el usuario.
Transform	Propiedad de los objetos de Unity para poder modificarlos.
Objeto	Unidad mínima para poder trabajar en Unity.
3D	Tres dimensiones.
Collider	Un collider es la estructura que hace sólidos a los objetos pudiendo controlar las colisiones entre ellos.
Partículas	Técnica de computación gráfica que permite mediante el uso de pequeños elementos crear fenómenos.
Monodeveloper	Entorno de desarrollo gratuito integrado en Unity para trabajar con scripts.
3Ds Max	Programa de creación de gráficos y animación en tres dimensiones.
SDK	Kit de desarrollo de software. Conjunto de herramientas para crear un software
License Key	Licencia para poder trabajar con realidad aumentada en la plataforma Vuforia.
AR Camera	Cámara de Vuforia para poder generar el efecto de realidad aumentada

Asset Store	Tienda virtual de Unity para poder descargar plugins en los proyectos.
Plugin	Complemento que se asocia a otros elementos para crear funcionalidades o comportamientos nuevos.
Rigibody	Componente que otorga a los objetos propiedades físicas
UI	Interfaz de usuario. Creado para que el usuario pueda interactuar con la aplicación en todo momento. Puede contener imágenes, botones o textos.

Bibliografía

[1] Plataforma Unity 3D. https://unity3d.com/es [Útimo acceso Septiembre 2016]

[2] Android y su evolución. https://es.wikipedia.org/wiki/Android [Útimo acceso Septiembre 2016]

[3] Source 2 Engine. http://www.valvesoftware.com/news/?id=16000 [Útimo acceso Septiembre 2016]

[4] CryEngine. https://www.cryengine.com/ [Útimo acceso Septiembre 2016]

[5] Unreal Engine 4. https://www.unrealengine.com/what-is-unreal-engine-4 [Útimo acceso Septiembre 2016]

[6] Historia de los videojuegos. http://www.fib.upc.edu/retro-informatica/historia/videojocs.html [Útimo acceso Septiembre 2016]

[7] Tipos de videojuegos. https://es.wikipedia.org/wiki/G%C3%A9nero_de_videojuegos#Tipos_de_videojuegos [Útimo acceso Septiembre 2016]

[8] Realidad aumentada: https://es.wikipedia.org/wiki/Realidad_aumentada [Útimo acceso Septiembre 2016]

[9] Plataforma Vuforia. https://developer.vuforia.com/ [Útimo acceso Septiembre 2016]

[10] Plataforma 3D Studio Max. http://www.autodesk.es/products/3ds-max/overview [Útimo acceso Agosto 2016]

[11] Video tutorial de 3Ds MAX 2017 para principiantes. https://www.youtube.com/watch?v=3oLXZB9fg6A [Útimo acceso Agosto 2016]

[12] Video tutorial para creación de texturas en Photoshop. https://www.youtube.com/watch?v=mcfxcUCBsek [Útimo acceso Agosto 2016]

[13] Documentación Unity. https://docs.unity3d.com/es/current/Manual/index.html [Útimo acceso Septiembre 2016]

[14] Tutorial Básico Unity. http://trinit.es/unity/documentacion%20proyectos/lecturas/Tutorial%20Unity3D%20-%20Franz%20Huanay.pdf [Útimo acceso Agosto 2016]

[15] Comunidad Unity 3D. http://www.unityspain.com/ [Útimo acceso Septiembre 2016]

[16] Scripting para Unity. https://unity3d.com/es/learn/tutorials/topics/scripting [Útimo acceso Septiembre 2016]

[17] Tutorial de C# para Unity. http://catlikecoding.com/unity/tutorials/ [Útimo acceso Agosto 2016]

[18] Manual C#. http://www.lawebdelprogramador.com/cursos/C-sharp/4428-Tutorial-C.html [Útimo acceso Agosto 2016]

[19] CABALLERO ROLDÁN, Rafael, CEREZO LÓPEZ, Yolanda, PEÑALBA RODRÍGUEZ, Olga, *Iniciación a la programación en C#: un enfoque práctico.* Delta publicaciones, 2006. ISBN-13: 978-8496477537

[20] LAVIERI, Edward, *Getting Started with Unity 5*. Packt Publishing Limited. Birmingham, 2015. ISBN-13: 978-1784398316